Gym Management System Project Report

Contents

[1. Introduction 1](#_Toc207706307)

[1.1 Project Overview 1](#_Toc207706308)

[1.2 Objectives 2](#_Toc207706309)

[1.3 Scope 2](#_Toc207706310)

[2. System Architecture 2](#_Toc207706311)

[2.1 High Level Design 2](#_Toc207706312)

[2.2 Components 3](#_Toc207706313)

[3. Technologies Used 3](#_Toc207706314)

[4. Features and Implementation 4](#_Toc207706315)

[4.1 Authentication and Authorization 4](#_Toc207706316)

[4.2 Admin Features 4](#_Toc207706317)

[4.3 Member Features 5](#_Toc207706318)

[4.4 Trainer (User) Features 5](#_Toc207706319)

[4.5 Common Features 5](#_Toc207706320)

[4.6 UI/UX 5](#_Toc207706321)

[5. Limitations and Potential Improvements 6](#_Toc207706322)

[6. Conclusion 6](#_Toc207706323)

# Introduction

## 1.1 Project Overview

The Gym Management System is a web based application designed to streamline gym operations, including member management, billing, notifications, supplement inventory, and diet planning. It supports three user roles: Admin (for overall management), Member (for viewing personal details), and User (Trainer, for viewing and searching member records). The system leverages Firebase for authentication and data storage, ensuring secure and realtime data handling.

The application consists of several HTML pages for user interfaces, a shared JavaScript file for logic and Firebase integration, and a CSS file for styling. It emphasizes role-based access control, basic CRUD operations, and simple reporting features.

## 1.2 Objectives

* Provide a secure login/signup system with role based redirection.
* Enable admin to manage members, create bills, send notifications, add supplements, and assign diets.
* Allow members to view their bills and notifications.
* Permit trainers (users) to view and search member details.
* Ensure a responsive, user friendly interface with error handling and feedback messages.
* Support basic reporting through CSV exports.

## 1.3 Scope

The system is a client-side application with Firebase backend integration. It handles user authentication, data storage, and basic queries but does not include advanced features like payment processing, realtime push notifications, or server-side validation.

# 2. System Architecture

## 2.1 High Level Design

* Frontend: HTML for structure, CSS for styling, and JavaScript for interactivity and Firebase interactions.
* Backend: Firebase Authentication for user management and Firestore for NoSQL data storage (collections like "users", "bills", "notifications", "supplements", "diets").
* Data Flow:
* Users interact via forms and buttons on dashboards.
* JavaScript functions handle events, authenticate users, and perform Firestore operations (e.g., add, update, query).
* Role checks ensure access control, redirecting unauthorized users to the login page.

## 2.2 Components

* Pages:
  + index.html: Login page with email/password fields and links to signup/reset password.
  + signup.html: Signup form with name, email, password, and role selection (member, user, admin).
  + admin\_dashboard.html: Admin interface for adding members, creating bills, sending notifications, exporting reports, managing supplements, and assigning diets.
  + member\_dashboard.html: Member view for bill receipts and notifications, with a refresh button.
  + user\_dashboard.html: Trainer dashboard for viewing member details and searching records.
* Shared Files:
  + script.js: Core logic including Firebase config, auth functions, data loading, and CRUD operations.
  + style.css: Global styling for a clean, responsive UI.

# 3. Technologies Used

* Frontend:
* HTML5: For page structure and forms.
* CSS3: For responsive design, including media queries for mobile devices.
* JavaScript (ES6+): For DOM manipulation, event handling, and asynchronous operations (async/await with Promises).
* Backend/Database:
* Firebase Authentication: Email/password-based auth with role based access.
* Firestore: NoSQL database for storing user data, bills, notifications, supplements, and diets.
* Libraries:
* Firebase SDK: Integrated directly in script.js for auth and database operations.
* Development Notes:
* No additional frameworks (e.g., React or Vue) are used, keeping it lightweight.
* Client-side only; no server-side code (e.g., Node.js).

# 4. Features and Implementation

## 4.1 Authentication and Authorization

* Signup (signupUser): Creates a new user in Firebase Auth and stores details (name, email, role, active status, joinDate) in Firestore. Role selection is validated to prevent invalid entries.
* Login (loginUser): Authenticates users and redirects based on role fetched from Firestore.
* Logout (logoutUser): Signs out and redirects to login.
* Role Check (checkAuthAndRole): Runs on page load to verify user and role, signing out unauthorized users.
* Password Reset (forgetPassword): Sends reset email via Firebase.
* Security: Uses Firebase's built in security. Admin actions like adding members require re-authentication as a client-side workaround.

## 4.2 Admin Features

* Member Management:
* Add Member (addMember): Form for name, email, password, join date, fee package. Creates user in Auth and Firestore.
* Edit Member (updateMember): Placeholder function (alert); needs extension for full implementation.
* Delete Member (deleteMember): Soft delete by setting active: false in Firestore.
* List Members (loadMembers): Fetches active members and displays with edit/delete buttons.
* Billing (createBill): Form to select member, enter amount and date; stores in "bills" collection.
* Notifications (assignNotification): Form to select member and enter message; stores in "notifications" collection.
* Supplement Store (addSupplement): Form for name, price, quantity; stores in "supplements" collection. Lists existing supplements.
* Diet Details (addDiet): Form to select member and enter diet plan; stores in "diets" collection. Lists assigned diets.
* Reporting (exportReport): Exports member data (name, email, join date, fee package) as CSV using client-side generation and download.

## 4.3 Member Features

* View Bills (loadBills): Fetches and displays bills for the loggedin member from "bills" collection.
* View Notifications (loadNotifications): Fetches and displays notifications for the member from "notifications" collection.
* Refresh: Reloads the dashboard data.

## 4.4 Trainer (User) Features

* View Members (loadMembers): Displays active members without edit/delete options.
* Search Members (searchMembers): Client-side filtering by name using lowercase comparison.

## 4.5 Common Features

* Message Display (showMessage): Temporary success/error messages with color coding.
* Dropdown Population (loadMembersForSelect): Dynamically fills member selects for forms.
* Data Loading: Functions like loadSupplements and loadDiets fetch and display lists in admin dashboard.

## 4.6 UI/UX

* Styling: Clean, modern design with flex box for layout, rounded borders, and hover effects. Responsive for screens <600px.
* Forms: Use HTML5 validation (required fields) and JavaScript for submission handling.
* Lists: .listitem class for consistent display of data with space-between justification.

# 5. Limitations and Potential Improvements

* Security: Client-side user creation/deletion is insecure; recommend server-side Firebase Admin SDK for production.
* Scalability: Client-side filtering (e.g., search) may perform poorly with large datasets; implement Firestore queries for server-side filtering.
* Features:
* No realtime updates (e.g., live notifications); add Firebase Cloud Messaging.
* Basic validation; add password strength checks and input sanitization.
* No image uploads (e.g., for supplements or profiles).
* Hardcoded metrics in admin dashboard; integrate actual analytics.
* Error Handling: Basic try catch; enhance with more specific error messages.
* Testing: Not covered; add unit tests for JS functions.
* Accessibility: Basic; improve with ARIA attributes and keyboard navigation.
* Deployment: Assumes Firebase Hosting; ensure HTTPS for security.

# 6. Conclusion

The Gym Management System provides a functional prototype for gym operations with secure authentication and role based features. Built with simplicity in mind, it demonstrates effective use of Firebase for a fullstack web app without a traditional backend server. With the noted improvements, it can evolve into a production ready application. The code is well structured, with clear functions and comments, making it easy to extend.

Date of Report: September 02, 2025

Prepared By: Raman Sagar